Programming in Modern C++
Professor Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 11
Classes and Objects

Welcome to programming in modern C++. We are starting week 3 with module 11.

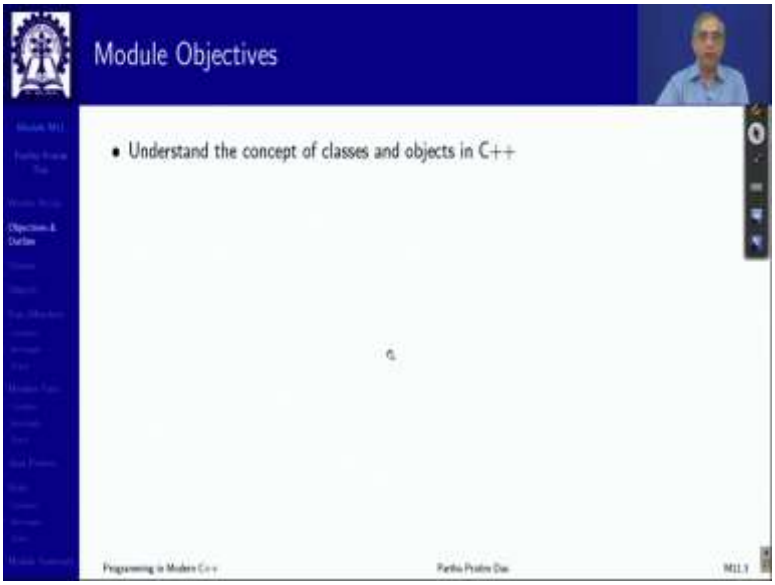
(Refer Slide Time: 0:37)

In the first week, we have looked at a variety of features, which are basically procedural features only. But those which make C++ a better C kind of language adding, for example, CV qualifiers, the concept of reference allowing us to do call by reference and return by reference, minimizing copies and solving a number of other problems. It allows us to do default parameters to default the values of parameters which are not frequently used.

And this is closely supported with the function overloading, which is known as static polymorphism. We have studied about the resolution mechanisms of overloading. And then specifically, we looked at the differences between operators and functions, and introduced the concept of operator functions and how using that operators can be overloaded.

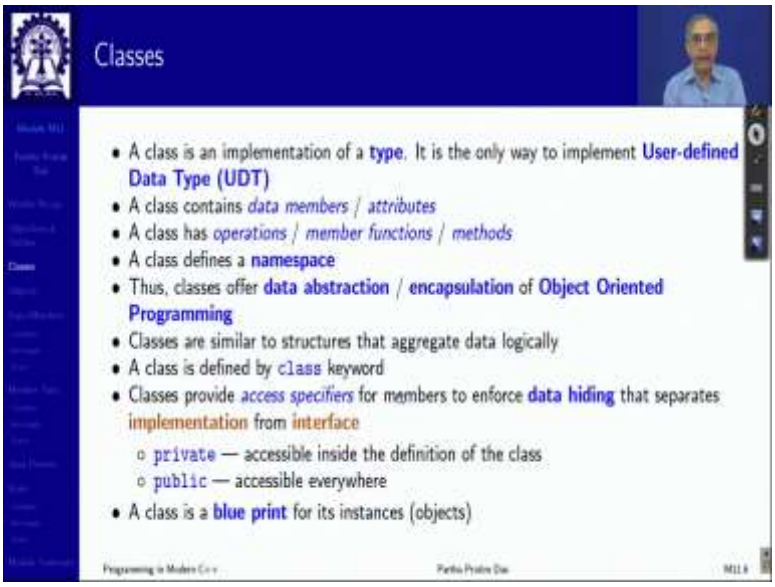
And we concluded the week with the discussions of dynamic memory management in C++, which is now very strongly enhanced from the C standard library style of malloc free and the dynamic memory management, new delete and its different variants actually becomes the part of the language itself. It is no more externally supported by the library.

(Refer Slide Time: 2:31)



So, we will move on from today to introduce C++ as a object oriented language. I am sure you all have heard about C++ in the context of object oriented programming. And that is one of the motivations you are attending this course. And for the first 2 weeks, we were just creating the foundation to discuss all this and from this module, we will actually introduce variety of object orientation concepts and how they are used, defined in C++. So, we will start with the basic concept of classes and objects and what are the ramifications of those.

(Refer Slide Time: 3:17)



So, we start with the classes. Now, here is a summary of what a classes, as we go through the examples, you will understand those in far more detail. So, class is as it is called is an implementation of a type we have built in types, we have derived types, we have typed certain types of type builders in C like struct, array, union and so on pointers as well. But here, we can actually be able to construct types, which had user defined called UDTs.

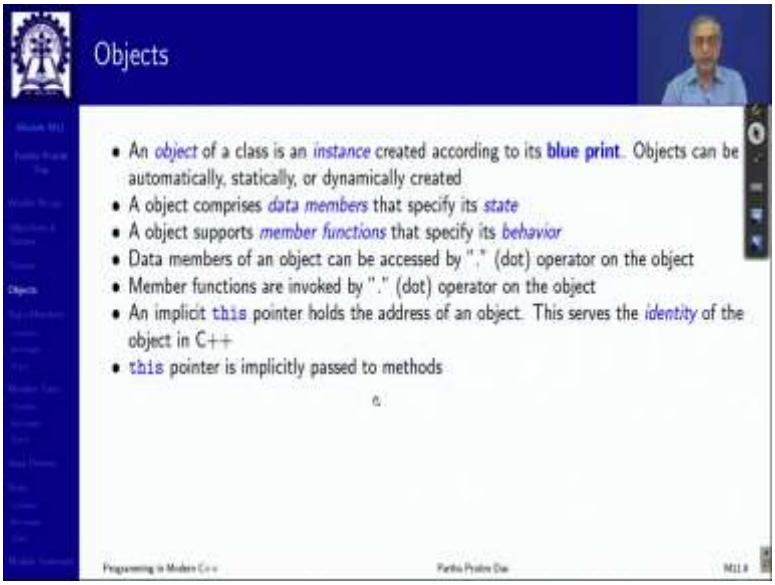
But the beauty would be that these types we can make them if we properly design them, we can make them exactly like behave exactly like a built in type and then keep on using them subsequently. So, naturally class is an aggregation. So, it has data members and attributes, it is also having a number of operations, member functions or method these terms are you know, interchangeably used to give the class its behavior.

So, you can think of a class being kind of a data structure as well. Classes a defines a namespace that is every symbol within the class is qualified by the name of the class we will talk about namespace per se later on in more detail. So, overall, the class offers data abstraction encapsulation, for object oriented programming. So, classes in a way are similar to structure but are substantially enhanced in C++.

So, it is defined by the class keyword and very important feature of a class is when we specify the data members and member functions, we can specify the access specification that is how the access of these data members and member functions can be done by anybody who is outside the class by private and public access specifiers.

And this gives us the basic the first level feature of data hiding or information hiding. So, in terms of the class, we achieve 2 basic objectives of object oriented programming, that is abstraction or encapsulation and the other is data hiding. So, class is a blueprint for its instances, which are called objects.

(Refer Slide Time: 6:07)

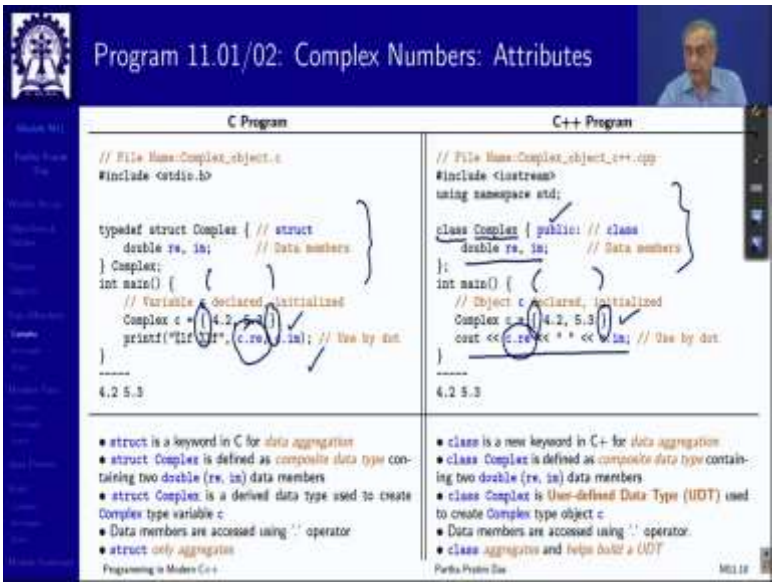


So, what is an object? Object is an instance of a class created according to the blueprint. So, these are the class says that these are the data members these are the member functions now, when you create the object, then the data members are specifically constructed and they can get different values which can keep on changing they are like variables.

And this collection of data members define the state of the object and the member functions which are common across objects will give its behavior, weak data members can be accessed by the dot operator you have seen in struct or union. Member function can also be invoked by the same operator and for every object, the identity of the object that is if I have 3 objects of the same class, then how do I differentiate them?

How does the system distinguish them? It is in C++ it is simply the address where the object resides. Therefore, that address identity becomes important for the programmer to know and it is available as a spatial pointer in every object called the this pointer and it is implicitly passed to every method or member function.

(Refer Slide Time: 7:34)



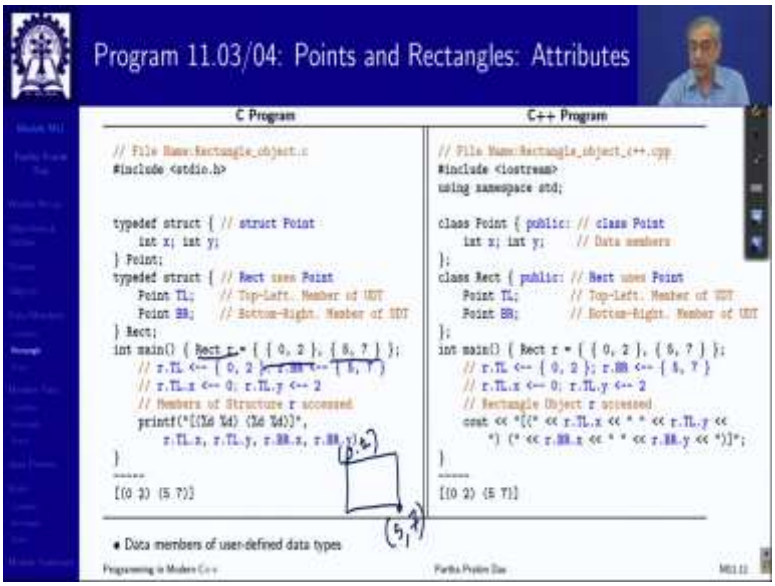
So, having said that, let me let us now dig into the different examples to look at. So, first start with the data members. So, we have on left a C program where we have alias a class a struct complex to represent complex numbers, we initialize them and then we print the values sample thing which you can always understand.

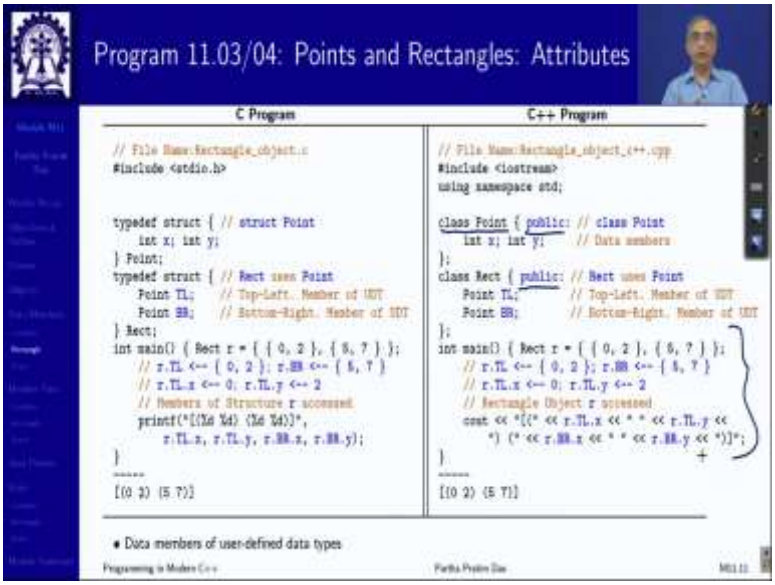
Please keep in mind that this style of initialization may not be supported in some of the compilers, if it does not, then you use the paranthesis only in place of the curly brace. So, on the right, the equivalent program in C++, where you specifically write class with class by itself becomes a type. So, it is not required to do any aliasing it is called complex and it has data members, which are re and im, the real and imaginary components of the complex number.

And this see as specifically written public here, the public means that any function which is not a part of the class can freely use it. So, we create an object like we did in C, but this object is now a proper object of the class complex and we write its components you can see the components are written in the same manner as you did in the case of structure. Again the point to note that, this style of initialization will not work in C++ 98 or 03.

So, if you are using C++ 90 or 03, use parenthesis in place of this. So, this is so, both class and structure provide the data aggregation they create a composite type with components and it kind of becomes in class it kind of becomes a user defined type and data members are data members can be accessed in the with the dot operator as you do in struct. And while struct only aggregates class aggregates and actually helps be build UDT, which will subsequently see.

(Refer Slide Time: 10:18)



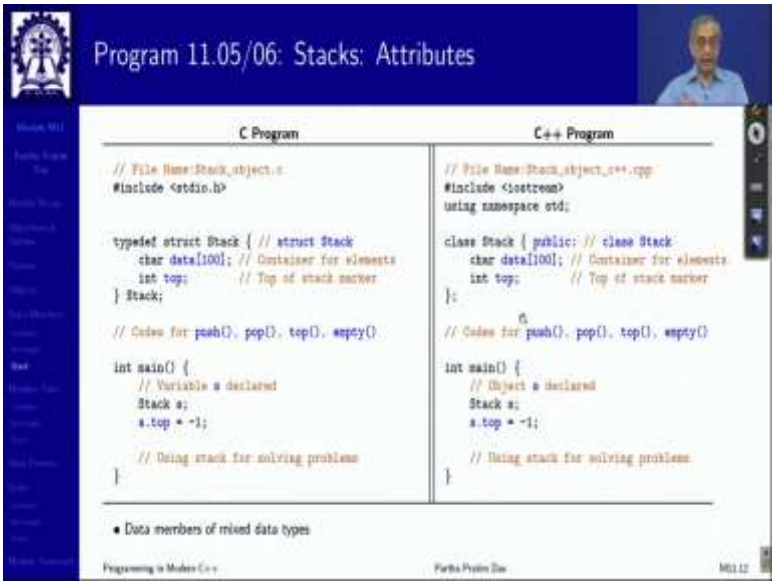


So, now, you have a number of examples following. So, I will not walk through each one of them, I have given a couple of examples, so, that you can get comfortable with the concept of data member here we are defining a point structure having 2 integer value so, it is kind of integer coordinate point and then I am defining another structure where I use the point structure to define 2 points of a rectangle top left and bottom right. Again you may have to change this initializer symbols to parenthesis.

So, with this what we are basically saying is created create a point 0, 2 create another point 5, 7 and top left and right bottom 0, 2, 5, 7 this rectangle is represented by the rect r. This is what is the C way of doing it in class it is very, very similar, you just use class you say these are public, in struct, that concept is not there, because instruct everything is public, in C everything is public.

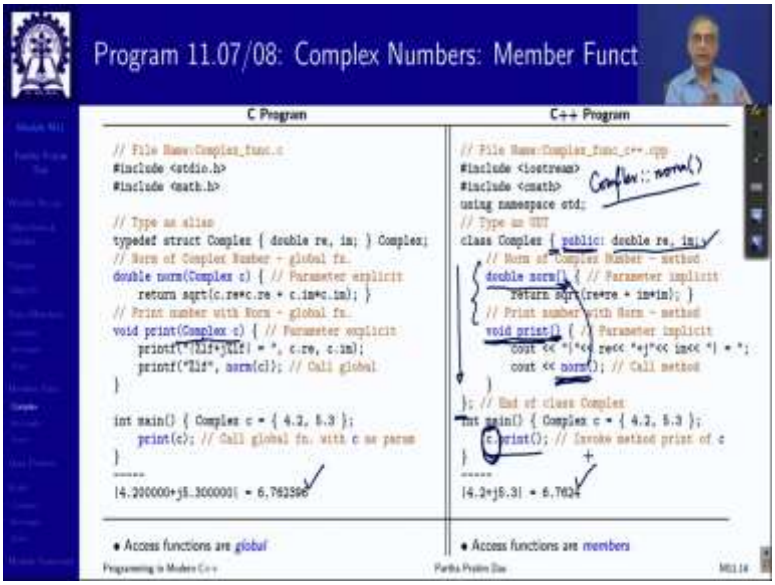
So, any program can directly any function can directly when change these members, but in a class you cannot do that only public ones you can. Similarly the rect and rest of the code is exactly same except for the cout stuff. So, this is so, you can the data members could be of built in type or they could be of types that you have already or classes that you have already created declare.

(Refer Slide Time: 12:18)



This is another for the stack, where you can easily see that you have a an array to store the stack values and a top to mark the top of the stack. So, details are easily understandable.

(Refer Slide Time: 12:37)



Now, let us introduce member functions. So, data members were very similar to the components of struct, but member functions are new in terms of the class only. So, what would you do in C if you have to deal with the complex structure? So, here you have the complex structure. And suppose I want to do 2 things.

One is I want to find the norm of this complex number, which is the square of the 2 components, add them and take square root that is a norm of the length of that complex vector. So, in if I have to do it in C, I have to define a function which is a global function, any function in C is global. And I have to tell the function that I am passing the a complex structure C, and then the necessary computation that is typical.

The other function that I want is to print those values. So, I have written a print routine, where also I pass the complex structure c and then I create an object and I print, you can see that print is actually using the other function norm to find the norm so, that you can print the value of the norm. So, this is this is the C way of doing it and which you all are absolutely familiar with.

In C++ we do something very, very interesting. This part is what you have already seen the data members. Now, what you do within the scope of the class, this is the scope of the class, you can see the pair of curly braces within that itself you can write a function. So, this known functional right within this. So, actually this known function is not global.

The actual name of the norm function is complex colon colon norm and this is the basic concept of having a namespace class as a namespace. So, any other class could also have a

norm function called norm. So, this is not global anymore. And so, since I have said this is public, so, going on words everything is public.

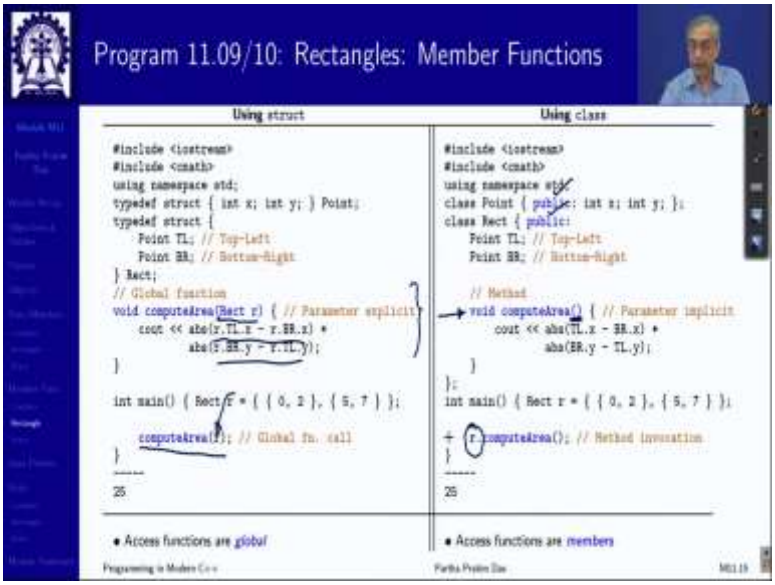
So, both these functions are also public, which means that any function outside of this class can also access them use them, and so does mean. Now, the difference here is that since norm and print are defined within complex, when I want to invoke it, I have to invoke it as a member function using the c dot notation, which is basically the this pointer or the identity of the c object the complex object c.

Now, if you do that, then you do not need to pass the struct complex as you were doing in C. So, this does not take any parameter, this does not take any parameter, because you will always invoke an a member function by the object. So, if you have a different object, you will say another complex d you will write d dot print, it will mean that print is for that d object not for the c object and that identification I will explain more is done through the this pointer.

So, if you look inside the print function now, you can see that print is actually calling norm and you can see here that it neither has a object reference nor it is passing anything why is it so, because when print can call norm only when print is in execution. If print is in execution, it already knows the object. Therefore, when I say norm within that, it necessarily means the same object on which print is currently working.

So, I do not either need to do an object reference there nor I need to pass anything this will automatically call norm for the same set of data members and do the computation and give the result back. You can see that we have these 2 results, the numbers look different, I had explained earlier because the default formatting of double are different between a C library and C++ string, you can change that formatting and make that the uniform.

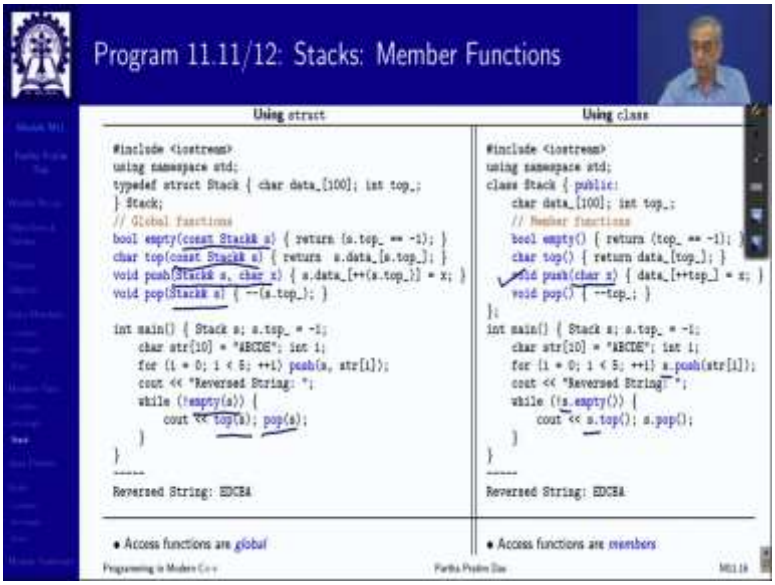
(Refer Slide Time: 17:47)



Here you have the member functions for the rectangle class, so you have the earlier types point and rect and you want to compute the area, so you declare it, global declare a global function and take the x difference, take the y difference, stick them in absolute because you do not know whether it is in the positive quadrant or it is one of them or both of them are in negative quadrant.

So, you take their absolute difference and compute and you have to pass this object here, this structure here. In C++, we have seen these other data members, which are public. So, also this member function, which I now introduce here, and like before, I do not need to pass any object because it will be invoked only with that object as we had seen before r dot compute area will do the same thing.

(Refer Slide Time: 19:03)

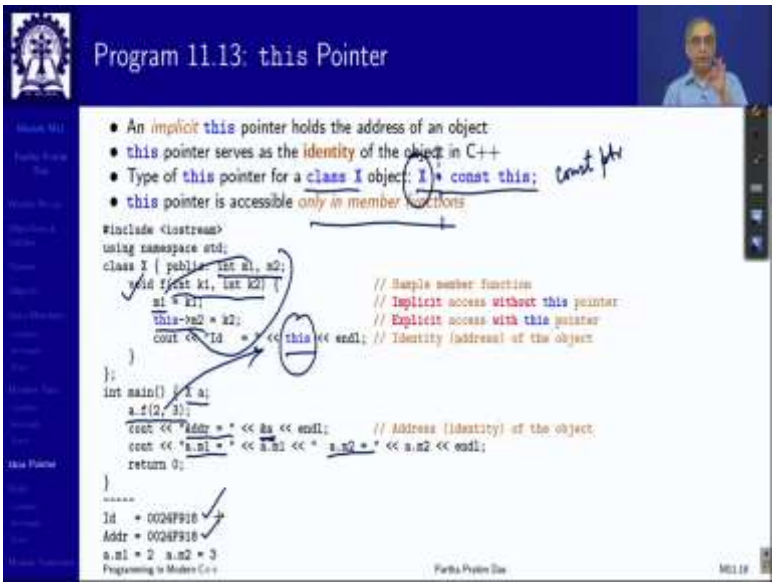


So, this is the role of the member functions in stack you have in C you have 4 global functions, the 4 basic stack data structure operations. In C++ class, you will have 4 member functions, all of them will have to be public so that you can access them anytime and write the same code with it with the difference of using them as the object dot notation instead of having to pass them all the time.

So, you can understand that if you pass the object, then you will have a lot more of overhead, maybe, of course to take care of that here in C++ I have taken these as reference and you can see that in 2 of them the stack is a constant reference because they do not change the stack. Whereas push and pop changes the stack so they are non-constant reference.

Here, you do not need to do any of this in push, you just need to tell what is the element that you want to push, because the scat is the object on which a specific instance of the object on which push has been invoked. So, this is how data members member functions can be used and they can freely use the data members if they are public they can be called by any other function where outside the class like main is doing here.

(Refer Slide Time: 20:36)



So, a little bit about the this pointer, I have already said that this pointer is the address of the object in the memory and it is an implicit pointer you do not need to define it, but it serves as the identity of the object there is no separate identity mechanism in C++ like Java has now, the it is important to note that if I have a class x then the type of the this pointer for any object of this class is x star const this.

The mechanism drive vertical line to the star it is on the pointed side. So, it says that it is it const pointer you can very well understand why is a const pointer because it is the address where the object is. So, obviously, as long as the object is there in memory, you cannot change that point. If you change that pointer then everything will go haywire, but there is no const on this site.

So, the object is not constant, you can make changes to there and this pointer is referring to the object. So, it is accessible only in member functions, it is not accessible by anyone else from outside. So, we have a class x here, which where I have 2 data members m1, m2 in the function of past 2 parameters k1, 2k.

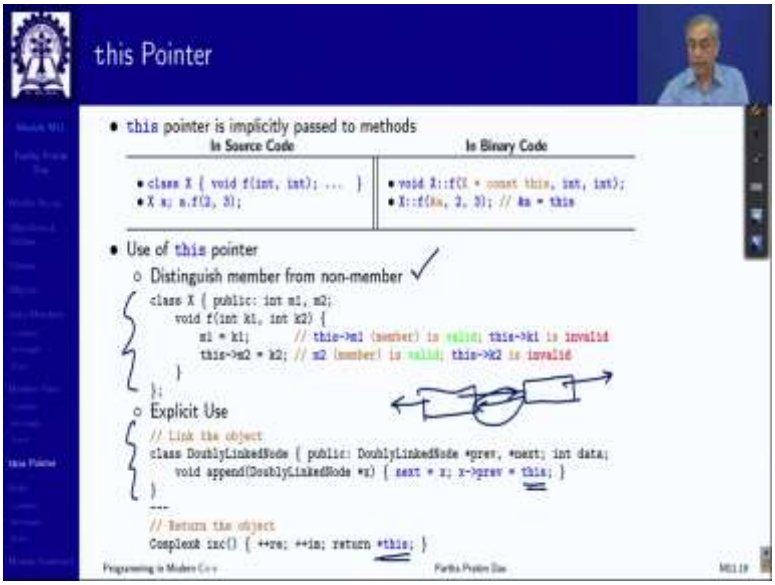
Now, within the function, which is a member function, I can access the data member either directly as m1 or I can access it is this pointer m1, m2 like this, any of these are allowed. In some cases, we explicitly use this pointer to make things clearer. But it is because writing this pointer also clutters the code and it is a lot of typing you have to do.

But so it is allowed that within the context of the member function, you can omit this mention of this pointer it implicitly means the current object. And you can directly use data member names or the member for other member function names to refer to them directly. And here I have done a print of this so that you can see.

And in main, we have created an object. And we have invoked this function f, a dot f with 2 and three. So, it does this printing, it prints the address, which is the address of a ampersand a you can always print that address, it prints the components. And with that, you can see that the Id that you have seen from f. And what you print here as address will be identical.

So, the Id this point in which I printed from f and the address that I have printed from main from outside, just taking it any variable I can take the address off. So, by doing that they are identical. So, that tells you that this pointer is necessarily the memory location where the object resides.

(Refer Slide Time: 24:10)



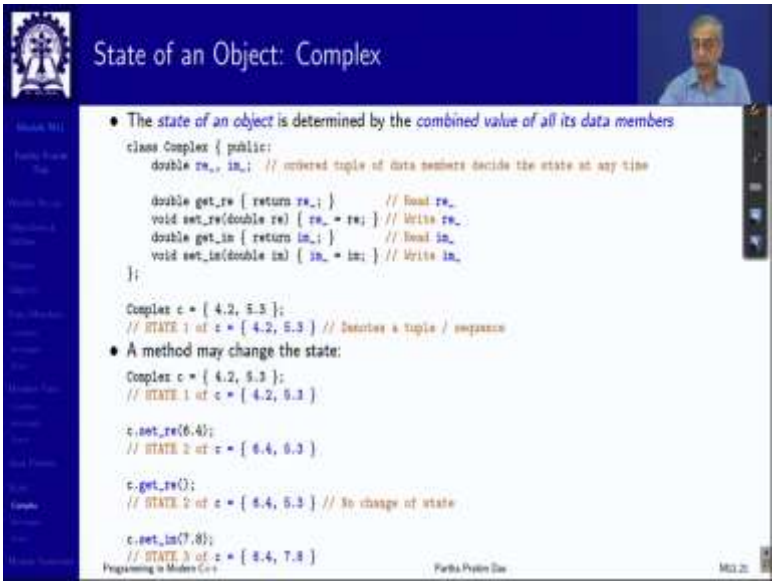
So, this pointer is implicitly passed to the methods, so we what it means that if I when I am writing a.f(2, 3). Actually, internally, though the function I have written is f intent, there are 2 integers. Internally, the compiler puts a first or you can set 0 width parameter which is at this point, the signature of this pointer.

So, when I call a.f(2, 3), the compiler actually translate it into taking the address of a and passing it as the 0th parameter. So, that is how actually that that is actual mechanism by which there is no magic but the by this mechanism, the member function is getting to know exactly which object to deal with. So, any anything that it is doing with the components, we had the data members, we had m1 and m2 are dereferenced from this point.

So, this pointer can you can be used to distinguish members from non-members as you have already seen, and in some cases, you really need to use it because without that, it is not possible to write the code for example, if you are trying to write the code for a doubly linked list like this then when you are actually read at a particular note, you need to set the previous node with its value.

So, with its pointer, so, you need to use this pointer, it will also be required at times to return an object from a function from a member function that is the object that you have, you want to return that same object back, we will see the actual context of why you need this and you will need to refer to the this pointer. So, in this it was is more an optional one, but these are explicit use without which you would not be able to write the code in the proper manner.

(Refer Slide Time: 26:34)

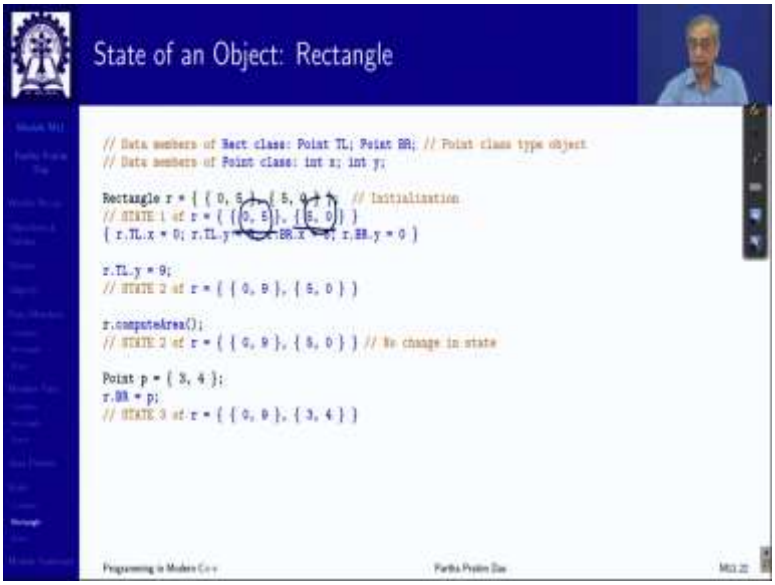


Now, before I close let me just tell you what is the state of an object? The state of an object is the collection of values of all its data members. So, all data members will have some value they can keep on changing, if the value changes, the object gets a different state. Mind you do not confuse between having 2 objects, which at any point of time have the same values of all the data members, we are not talking about that.

We are talking about the state of individual object, you can think about the state of a single variable, the state of a single variable is its value, it keeps on changing. So, it goes through different states here, it is an aggregation. So, the state is basically the collection of those values. So, if I talk about the double, there are 2 components.

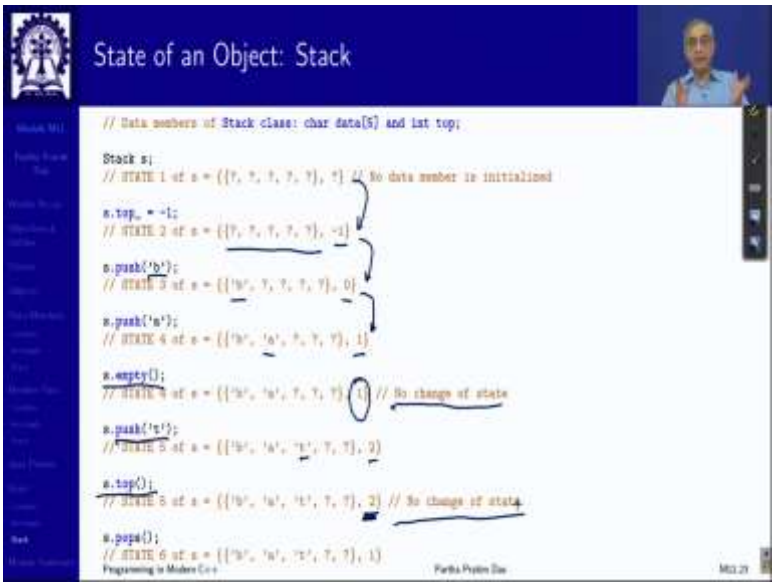
So, your state is always a pair of double values, as it keeps on changing because of different member functions operating on them or if these are public, so, somebody can come and access and directly change them. So, they will keep on changing accordingly. And he will have different states.

(Refer Slide Time: 27:56)



For a rectangle, these are the kinds of states you will have, because a rectangle has 2 points, every point has 2 integer coordinates. So, it is basically a pair of pairs of integers. It is a pair here for the 2 points left top and right bottom and each one is a pair. So, this double of doublet, doublet of tablets is basically the state and as you do different operation the state keeps on changing. We will often need to talk about the state of an object.

(Refer Slide Time: 28:38)



Similarly, here is a state of the stack. So, if the stack has a container, which is of size 5 characters, and top is the market, these are the 2 data members. So, it is the all values in that array and the top market value of the top marker besides the state, and it is possible that some of those are unspecified indeterminate because when you start there is no element in that

array. So, you do not know what so they are just question mark that is undefined. Similarly, top has not been defined.

Now, once you define tarp to be minus 1, then your array remains all undefined, but your tarp has become in a specific value. So, this is a different state from the state. Now you do a push. So, the top changes and the value suppose you are pushed you are pushed b, so the value comes in at the 0 at location. So, the array has changed the top also has changed the stack is in a different state.

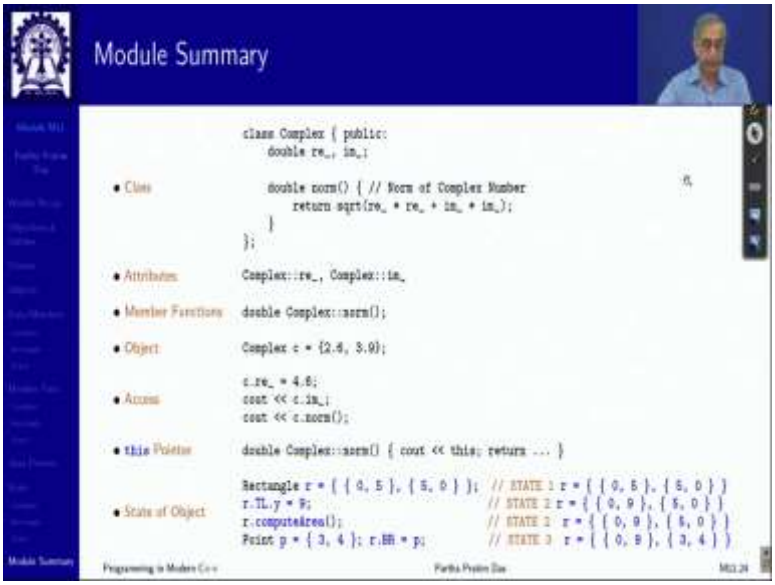
You push another then you have a introduced in the array as well as the top marker has changed. You have checked for s empty, so s empty is just checking if the stack is empty that is if top is minus 1 or not. So, it is not changing the state. So, it is it has to be kept in mind that and this notion will be used very frequently that whenever we use a member function, we are not necessarily changing the state.

Some member functions will change the state, some member functions will not change the state, they will just maybe do some checks there tests when we prints and so on, but they does not do not change the state. So there is no change of state with the empty here. Then again, we push, so t and we get here, these are the changes, so it is in a new state, then again, we check top, we get the value of top.

So, you have seen earlier in the reverse string, and so on. So, when I get that I am just reading this value of top. And I am not doing anything changing in the array or in the top. So, there is again, there is no change of state, so these empty and top these are the 2 which do not change the state. And you will remember that when I did the global function style, I passed these to these 2 functions, I passed the stack as a non-constant reference.

I am sorry, as a constant reference. Whereas for push and pop, I had to pass them as non-constant reference because they change the state. So, that is a basic idea.

(Refer Slide Time: 31:35)



So, this is the basic notions that we have introduced the class the attributes are data members, member functions object, the access mechanism, this pointer and the state of the object, very foundational notion about the whole of object oriented programming starting in C++. I hope you enjoyed that. And this will be extremely useful going forward. Thank you for your attention, and we will meet in the next module.